Long Short Term Memory Networks For Light Field View Synthesis

Matthieu Hog, Neus Sabater, Christine Guillemot

LONG SHORT TERM MEMORY NETWORKS FOR LIGHT FIELD VIEW SYNTHESIS

Matthieu Hog[1,2]*, Neus Sabater*[1]*, Christine Guillemot*[2]

[1] Technicolor R&I, [2] INRIA , Rennes, France

ABSTRACT

Because light field devices have a limited angular resolution, artificially reconstructing intermediate views is an interesting task. In this work, we propose a novel way to solve this problem using deep learning. In particular, the use of Long Short Term Memory Networks on a plane sweep volume is proposed. The approach has the advantage of having very few parameters and can be run on sequences with arbitrary length. We show that our approach yields results that are competitive with the state-of-the-art for dense light fields. Experimental results also show promising results with light fields with wider baselines.

Index Terms— Light Field, View Synthesis, Deep Learning

1. INTRODUCTION

Real light fields capturing devices exhibit different spatial and angular trade-offs. Plenoptic cameras trade spatial resolution against angular resolution, whereas camera arrays can only capture a limited set of views. Both depth image-based rendering (DIBR) and deep learning approaches have been proposed to increase angular resolution via view synthesis.

In classical DIBR approaches, scene depth is first estimated and used to warp and combine the input views. This has been done using user-defined image labeling [1] or superpixels [2]. However, methods relying on a hard depth prior to render an image [3, 4, 5] have the problem that faulty depth estimation yields very unpleasant artifacts in the rendered image. Other methods focus on formulating view synthesis as a maximum a posteriori estimation [6, 7]. A more recent approach [8] proposes to propagate depth uncertainties from depth estimation to image rendering. The results are numerically and visually superior to prior methods but it has some serious computational and memory issues.

In the last few years, deep learning approaches have been proposed for view synthesis. The authors in [9] propose a deep learning approach for light fields view synthesis. A Plane Sweep Volume (PSV), *i.e.* a set of views warped with disparities in a given range, is used as input of two twin networks towers with shared weights. The first network learns masks on each slice of the PSV, selecting zones that correspond to actual points in the scene. The second network learns the most probable color at a given point. The final image is rendered, as the linear combination of the two networks. Although the results are visually stunning and work perfectly for sparsely sampled views, the major problem of this approach is its complexity and the number of parameters. Because of this, the authors in [10] propose an approach tailored for narrow baseline light fields captured with plenoptic cameras. In this approach, the mean and variance of each slice of the PSV are fed to a first CNN, that learns a disparity on the synthetic view coordinate system. The corner images are then warped using the disparity and concatenated with the depth plus the new view index and passed into a second CNN that learns to refine and combine the warped corner images. The approach gives very good results on Lytro images, however the PSV is rather dense (100 slices), yielding a lot of operations on the first layers of the network. Besides, the network is trained for a fixed amount of slices and, as in [9], it requires to load the entire network at once for the forward pass. This idea of light field angular super-resolution has recently been improved (eg. using microlens images [11]), but with similar limitations.

In addition to the very high number of parameters which is an issue for computationally and memory limited devices, all current architectures are, by design, bound to only one disparity range. The features trained for a given device cannot be re-used for devices with different baselines. To solve these problems, in this paper we propose to use Recurrent Neural Networks (RNNs), and in particular Long Short Term Memory (LSTM) RNNs, to directly learn view synthesis from a PSV in a modular fashion with low complexity.

RNNs are neural networks in which instead of statically chaining layers, recurrent connections are made between the so-called *cells* with the particularity that all cells share its parameters. Each cell takes as input the output of the previous cell and a different element of a sequence and outputs a value and provides some information to the next cell. Such network structure has several advantages. First and foremost the number of parameters is very small compared with CNNs, since the cells have the same weights. Second, by changing the number of cells, the RNN can be used with different sequence lengths. Also, the recurrent loop can be run sequentially during the forward pass, making the GPU memory requirement a lot smaller than a conventional CNN.

We show that our approach can produce images of comparable quality with state-of-the-art methods with significantly

lower number of weights. As an interesting byproduct, we also show that RNNs somehow learn differently than CNNs for view synthesis. Indeed, while a single CNN usually gives blurred results when used for view synthesis, the proposed approach does not suffer significantly from this defect. While the approach is assessed with dense light fields, we show that the approach can be extended to wider baselines.

2. LSTMS FOR VIEW SYNTHESIS

LSTMs [12] networks are a variation of RNNs where each cell has a *memory*, a *state*, that is updated across the RNN loop. This is to help the propagation of information across RNN iterations. It also solves the problem of vanishing gradients [13]. Classical LSTMs are made of 3 parts, composed of neural layers activated by a sigmoid function, called *gates*. In particular, we denote C the cell memory, h the cell output. Because LSTMs are typically used to process temporal sequences (*e.g.* text, sound), a RNN cell is run on temporal steps noted t. The cell state is updated from C_{t-1} to C_t by removing information (with a point-wise multiplication), then by adding new information (with an addition step). This is done with an input gate σ^i, that filters relevant information in the previous cell output and the current input, an update layer $Tanh^u$ and gate σ^u, that computes and selects the update to be done to the cell state, and finally an output gate σ^o to select relevant feature to pass in the RNN loop. Specifically, because we are dealing with images, we use convolutional LSTMs [14], a variant of LSTMs that replaces the matrix multiplication done at each layer with convolutions, making the LSTMs cell effectively capturing both spatial and sequence information.

The main idea behind our proposal is that, instead of establishing links in the z-dimension of the PSV via a single, or multiple layers, we learn how to do it with a RNN. Fig. 1 offers a comprehensive overview of our approach. In particular, we try to learn directly the image to synthesize $\hat{I}_d$ from the PSV, without any depth or selection map estimation. Each LSTM cell takes as input the 4 corner views warped with a depth plane d and concatenated along the color dimension (the depth of the input is then 12). In other words, we treat the depth dimension of the PSV as a temporal dimension for LSTMs traditional approaches. As in other learning methods [10, 9] a groundtruth view is used to evaluate the model prediction and update the learned filter.

In our approach, the cell memory is used to encode the most probable color for each pixel of the new view. One could expect the LSTM to behave as follows. The input and update gates σ^i and σ^u will select the most relevant color features from the previous cell output and from the current PSV slice. The update gate σ^u will deduce, from the previous RNN iterations and the current PSV slice color values what color features are unlikely to compose the true new view colors. The update gate and layer σ^u and $Tanh^u$ will then compute the new color features to save, presumably because they are more likely. After the new cell memory state is updated, the output gate σ^o will filter out the color features that are important to pass on to the next cell.

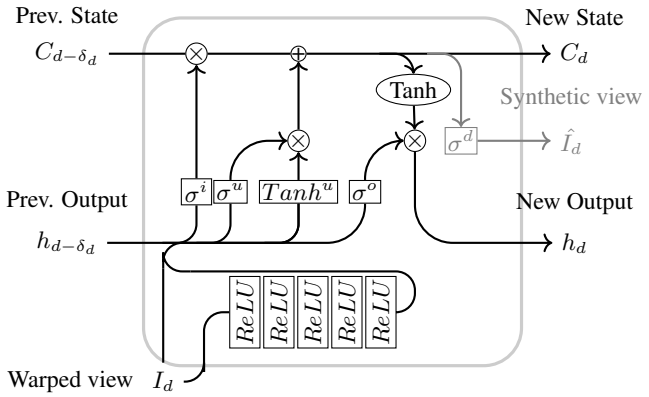

Fig. 1: The LSTM cell used in our approach. Each cell takes as input the 4 corner views warped at a specific depth plane d. It embeds a small CNN to learn features on each slice of the PSV that are later concatenated with the input. At the very last RNN iteration, we pass the cell state into a single layer (in gray) to generate the final image $\hat{I}_d$.

We use a fixed kernel size of 3×3 for all the gates and a feature size of 32. Using fewer filters has a negative impact on the approach performance and interestingly, adding more as well. On the very last RNN iteration, we pass the state of the cell through a simple convolutional layer σ^d with a kernel size of 1×1 and depth of 3 in order to *decode* the learned features into the final image. Interestingly, we have noticed that using a deeper *decoding* step provides drastically worse results. Since the gates of the LSTM are composed of only one layer, the learnt filters and update function are rather limited. In order to learn more complex representations of each slice of the PSV, we use a small CNN. In contrast to what has been done in the past [15], we learn the features on the set of 4 views and not on each view independently. Equally, we do not replace the cell input with the learned features but concatenate them. This is to introduce more context in-between views rather than simple image features, but without replacing the input signal, and in practice, we find out that it improves significantly the results justifying the added parameters. The used network is a simple CNN composed of 5 layers of 32-features with a kernel size of 3×3. We perform the RNN loop warping the views from foreground to background (as in the rendering step of [8]). Note that reversing the order provides a significantly worse result. Using a bi-directional scheme, as in [16], does not improve the results but also gives slightly worse results. We hypothesize this is due to the way the network deals with occlusions, performing an operation akin to *z-clipping* in the LSTM loop.

Because it has been shown to provide sharper image re-

Fig. 2: Visualization of the LSTM state at different RNN iterations (to be read from left to right). On top we show the decoded memory state of the LSTM, on the bottom, we display the refocused 4 input views to see what depth plane is currently valid.

sults [17], our loss function is the $L1$ difference between the groundtruth image I and the reconstructed image $\hat{I}_d$ at the very last iteration of the LSTM loop: $||\hat{I}_d - I||_1$.

3. EXPERIMENTS

We follow the training protocol proposed in [10]. That is to say, we use Lytro Illum subaperture images as training input and groundtruth. The central 7×7 views are extracted from the microlens images, as they do not suffer from vignetting and chromatic aberrations. The PSV is rendered for each integer disparity in the range $[-12, 12]$ for the central view. The groundtruth central view is used as a reference to compute the loss and the test metrics. In order to avoid color and optical aberrations that are specific to each camera, we augment the dataset of [10] with 3 other Lytro Illum datasets [18, 19, 20]. The network is trained using ADAM [21], with a learning rate of 0.0003, and for $200K$ iterations. We use a batch size of 10 and as in [10], we train our network on patches of size 128×128. We do not use batch norm and do not crop the final images borders, neither during training nor during testing. Tensorflow implementation of dynamic RNNs is used for training with our custom cell. The overall training time is slightly less than 2 days, although the results do not change much after $100K$ iterations.

Note that we refer to our supplementary material[1] for extra details that could not fit in the paper.

Model Validation: In order to verify that the network learns as expected we employ the following strategy. We manually unroll the RNN loop and apply the *decoding* layer σ^d of the last iteration to the cell memory state at each iteration. Although it is not strictly what the LSTM memory saves, this technique gives us a visualization of what is happening from an iteration to another. On Fig. 2, we clearly see that zones that have been in focus in the previous iterations adopt their final color, overwriting the previous features, at a current time step. This is done independently at each scene depth plane, showing that our LSTM strategies do indeed learn features about the best color its has seen so far. This result is consistent across the entire test set. Also, we observe a correlation between the artifacts in the selection done at each step and artifacts in the final image. These artifacts mostly occur at the edge of objects (as in the third slice of Fig. 2) and usually yield blurred borders in the final image.

[1] `http://www.irisa.fr/temics/demos/LSTMViewSynthesis/index.html`

(a) Synthetic view [10] (b) Synthetic view LSTM

(c) Leaf [10] (d) Leaf LSTM (e) Petal [10] (f) Petal LSTM

Fig. 3: Comparison with [10]. (a&b) show the entire image. In (c&d) we observe that our approach suffer less at objects boundaries. However, as shown in (e&f), our approach is subjects to color bleeding artifacts is some areas.

Quantitative Comparisons: The implementation of [10] (done by its authors) is used as a baseline to evaluate our method. We also use their test set composed of 30 Lytro Illum images. Note that, as in their experiments, the interpolation and metrics are computed on the input under-exposed images, but gamma-corrected images are shown. For the sake of comparison, we also re-implemented and re-trained the frame interpolation method in [22] adapting slightly for the view synthesis case. This method uses a deep, hierarchical network, estimating an optical flow residual at 3 different scales. Details about the modifications can be found in the supplementary.

Fig. 4: Visualization of the LSTM memory state for a wide baseline dataset.

Table 1: Image quality metrics for the compared approaches

Dataset	[22]		LSTM		[10]	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Cars	32.87	0.96	30.24	0.97	31.93	0.97
Flower1	31.36	0.97	34.21	0.97	33.43	0.97
Flower2	31.43	0.97	33.26	0.97	32.11	0.96
Rock	31.54	0.96	33.88	0.98	35.63	0.97
Seahorse	31.75	0.96	32.80	0.95	31.62	0.97
Average	34.51	0.96	38.03	0.96	37.18	0.97

We compare the three approaches for all the images of the test set of [10]. We note that our approach is at least on par with [10], with an average PSNR of 38 vs 37.2 for [10] and 34.5 for [22]. Our approach performs worse than [10], but better than [22] in terms of MSE and SSIM. Tab. 1 shows the results for the examples in [10] and the average for the entire dataset. The full table of results is included in the supplementary. The main reason why the two full-resolution images are numerically and visually better than the hierarchical is that they rely on full-resolution images for the entire pipeline. Small details are not well captured in the depth estimation, and are not recovered during the upper scale depth estimation either, since the depth estimation is hierarchical. We show an example of this in the supplementary material. Visually, it is hard to distinguish the results from the two approaches. We show a visual comparison in Fig. 3, note that in [10], an important portion of the image is cropped. For both approaches, most of the error is contained in the objects boundaries. Our approach suffers slightly less from these artifacts (*e.g.* with the foreground leaf) but can sometimes introduce slight color bleeding, in some objects (*e.g.* the purple flower petal on the white wall). This could be explained by the fact that our network has to reconstruct colors, while [10] warps the input textures. In terms of parameters, [10] has 1 644 204 while ours only has 114 400.

Towards variable baseline view synthesis: as stated in the introduction, one of the interest of LSTMs is that they can be used on sequences with variable length. As a last experiment, we test how generalizable is the network trained on small disparities to a wide disparity setup. We use the *Beergarten* sequence in [23], as it is close to a Lytro sequence (*i.e.* a rectified light field with the same vertical and horizontal baseline). The PSV is composed of 43 slices against 25 for the Lytro case. On Fig. 4 we notice that the results are not as good as for the densely sampled dataset, but the network exhibits the same behavior at each iteration, even if it has not trained for this kind of data.

4. CONCLUSION

Our experiments suggest that RNNs with the proposed architecture are suitable for view synthesis. Not only they provide synthetic views with competitive quality with the state-of-the-art, but also with a reduced number of parameters. However, our experiments focused on generating the central view. Since the input of the method is a PSV, we do not expect the results of a training on arbitrary views, done for instance by inputting the PSV warped in the new view coordinate system and concatenated with normalized view index, to be drastically significant. We also show that the approach can be used with light fields with wider baselines despite not being trained explicitly with this particular data. We believe that a LSTM network trained with a dataset composed of both sparse and dense corner views would be able to perform well on arbitrary baselines.

They are numerous improvements that can be brought to this method. First, the use of deeper gates (*i.e.* gates with more than one layer) would allow the LSTM cells to learn more complex representation about the best synthetic view color. This might be necessary to extend the approach to wider baselines and multi-view rendering. Another improvement could be to follow classical approaches and decompose the problem of view synthesis by learning depth. We could imagine for instance passing the normalized disparity plane as an extra channel in the cell input and instead of learning features about the most probable color, let the network infer the most probable depth of each pixel of the view to synthesize. The output would then be a disparity map and possibly interpolation weights as in [22], used to generate the final image. This would allow keeping a level of disparity range genericity in contrast to the refinement network in [10].

5. REFERENCES

[1] Gaurav Chaurasia, Olga Sorkine, and George Drettakis, "Silhouette-aware warping for image-based rendering,"

in *Computer Graphics Forum*. Wiley Online Library, 2011, vol. 30, pp. 1223–1232.

[2] Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis, "Depth synthesis and local warps for plausible image-based navigation," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 3, pp. 30, 2013.

[3] Leonard McMillan and Gary Bishop, "Plenoptic modeling: An image-based rendering system," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM, 1995, pp. 39–46.

[4] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen, "The lumigraph," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 43–54.

[5] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen, "Unstructured lumigraph rendering," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, pp. 425–432.

[6] Sven Wanner and Bastian Goldluecke, "Variational light field analysis for disparity estimation and super-resolution," *PAMI*, vol. 36, no. 3, pp. 606–619, 2014.

[7] Sergi Pujades, Frédéric Devernay, and Bastian Goldluecke, "Bayesian view synthesis and image-based rendering principles," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3906–3913.

[8] Eric Penner and Li Zhang, "Soft 3d reconstruction for view synthesis," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 6, pp. 235, 2017.

[9] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely, "Deepstereo: Learning to predict new views from the world's imagery," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5515–5524.

[10] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi, "Learning-Based View Synthesis for Light Field Cameras," *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2016)*, vol. 35, no. 6, 2016.

[11] M Shahzeb Khan Gul and Bahadir K Gunturk, "Spatial and angular resolution enhancement of light fields using convolutional neural networks," *IEEE Transactions on Image Processing*, vol. 27, no. 5, pp. 2146–2159, 2018.

[12] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al., "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," .

[14] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, 2015, pp. 802–810.

[15] Wenjie Luo, Alexander G Schwing, and Raquel Urtasun, "Efficient deep learning for stereo matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5695–5703.

[16] Mike Schuster and Kuldip K Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[17] Michael Mathieu, Camille Couprie, and Yann LeCun, "Deep multi-scale video prediction beyond mean square error," *arXiv preprint arXiv:1511.05440*, 2015.

[18] Raj Shah Abhilash Sunder Raj, Michael Lowney and Gordon Wetzstein, "Stanford lytro light field archive," `http://lightfields.stanford.edu/LF2016.html`, 2016.

[19] Martin Rerabek and Touradj Ebrahimi, "New light field image dataset," in *8th International Conference on Quality of Multimedia Experience (QoMEX)*, 2016, number EPFL-CONF-218363.

[20] Christine Guillemot Rodrigo Daudt, "Inria lytro illum light field dataset," `https://www.irisa.fr/temics/demos/IllumDatasetLF/index.html`, 2016.

[21] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] Joost van Amersfoort, Wenzhe Shi, Alejandro Acosta, Francisco Massa, Johannes Totz, Zehan Wang, and Jose Caballero, "Frame interpolation with multi-scale deep loss functions and generative adversarial networks," *arXiv preprint arXiv:1711.06045*, 2017.

[23] Lukasz Dabala, Matthias Ziegler, Piotr Didyk, Frederik Zilly, Joachim Keinert, Karol Myszkowski, Hans-Peter Seidel, Przemyslaw Rokita, and Tobias Ritschel, "Efficient Multi-image Correspondences for On-line Light Field Video Processing," *Comput. Graph. Forum*, vol. 35, no. 7, pp. 401–410, 2016.